Amendments to the Claims:

1. (original) An apparatus comprising a processor and a memory storing computer program code, wherein the memory and stored computer program code are configured, with the processor, to cause the apparatus to at least:

implement a scheduler incorporating an algorithm for ordering the running of threads of execution having different priorities, the scheduler maintaining a ready list of threads which are scheduled to run on the device, ordered by priority; and

implement at least one locking mechanism configured to block access to a resource from all threads except for a thread that holds the locking mechanism,

wherein the locking mechanism(s) comprise(s) a mutex including a pointer, the pointer included in the mutex pointing to the thread holding the mutex or being null if the mutex is free, the mutex further including a flag indicating whether or not the mutex is contested, and

wherein in an instance in which the scheduler selects a thread on the ready list to run, but the selected thread is blocked from running because a resource it requires is blocked, the scheduler does not switch to the blocked thread but retains the blocked thread in its place by priority on the ready list and instead yields to the thread which holds the locking mechanism and causes the thread which holds the locking mechanism to run.

2. (Previously Presented) An apparatus according to claim 1 wherein states are assigned to threads and the ready list comprises of all threads having a common state.

3. (Previously Presented) An apparatus according to claim 2 wherein a blocked thread is not permitted to change its state.

4. (Previously Presented) An apparatus according to claim 1 wherein the ready list is subdivided in accordance with the priority of the threads it contains.

5. (Previously Presented) An apparatus according to claim 1 wherein a thread contains a pointer to any locking mechanism it is blocked on.

6. (Previously Presented) An apparatus according to claim 1 wherein the memory and stored computer program code are configured, with the processor, to cause the apparatus to implement a plurality of non-nestable locking mechanisms.

7. (Previously Presented) An apparatus according to claim 1 wherein the memory and stored computer program code are configured, with the processor, to cause the apparatus to call the scheduler at the end of an interrupt service routine which is caused to run.

8. (Canceled)

9. (Currently Amended) An apparatus according to claim [[8]] 1 wherein the algorithm is configured to delegate memory management to a replaceable memory model configured in dependence upon a configuration of the apparatus.

10. (Previously Presented) An apparatus according to claim 9 wherein the memory model is configured to run in either pre-emptible or non-preemptible modes.

11. (Currently Amended) An apparatus according to claim 10 wherein [[a]] the mutex is configured to protect the module from running in the pre-emptible mode.

12. (Previously Presented) An apparatus according to claim 1 wherein the scheduler is included in a kernel of an operating system of the apparatus.

13. (Previously Presented) An apparatus according to claim 12 wherein the kernel comprises a microkernel or a nanokernel, and wherein the threads are, respectively, microkernel or nanokernel threads.

14. (Currently Amended) An apparatus according to claim 12 wherein the memory and stored computer program code are configured, with the processor, to cause the apparatus to call the scheduler ~~each time~~ in response to the kernel [[is]] being unlocked.

15. (Previously Presented) An apparatus according to claim 1, wherein the apparatus comprises a mobile computing device.

16. (Previously Presented) An apparatus according to claim 15, wherein the mobile computing device comprises a smart phone.

17. (Currently Amended) A method comprising:

providing a scheduler incorporating an algorithm for ordering the running of threads of execution having different priorities, the scheduler maintaining a ready list of threads which are scheduled to run on a computing device, ordered by priority; and

providing at least one locking mechanism configured to block access to a resource of the device from all threads except for a thread that holds the locking mechanism,

wherein the locking mechanism(s) comprise(s) a mutex including a pointer, the pointer included in the mutex pointing to the thread holding the mutex or being null if the mutex is free, the mutex further including a flag indicating whether or not the mutex is contested, and

wherein in an instance in which the scheduler selects a thread on the ready list to run, but the selected thread is blocked from running because a resource it requires is blocked, the scheduler does not switch to the blocked thread but retains the blocked thread in its place by priority on the ready list and instead yields to the thread that holds the locking mechanism and causes the thread that holds the locking mechanism to run.

18. (Currently Amended) A non-transitory computer-readable storage medium storing computer program code for an operating system for a computing device, the operating system comprising:

a scheduler incorporating an algorithm for ordering the running of threads of execution having different priorities, the scheduler configured to maintain a ready list of threads which are scheduled to run on the device, ordered by priority; and

at least one locking mechanism configured to block access to a resource of the device from all threads except for a thread that holds the locking mechanism,

wherein the locking mechanism(s) comprise(s) a mutex including a pointer, the pointer included in the mutex pointing to the thread holding the mutex or being null if the mutex is free, the mutex further including a flag indicating whether or not the mutex is contested, and

wherein in an instance in which the scheduler selects a thread on the ready list to run, but the selected thread is blocked from running because a resource it requires is blocked, the scheduler is configured to not switch to the blocked thread but retain the blocked thread in its place by priority on the ready list and instead yield to the thread that holds the locking mechanism and cause the thread that holds the locking mechanism to run.

19. (Previously Presented) A method according to claim 17 wherein states are assigned to threads and the list comprises of all threads having a common state.

20. (Previously Presented) A method according to claim 19, wherein a blocked thread is inhibited from changing its state.

21. (Previously Presented) A method according to claim 17, further comprising subdividing the ready list in accordance with the priority of the threads it contains.

22. (Previously Presented) A method according to claim 17 wherein a thread contains a pointer to any locking mechanism it is blocked on.

23. (Previously Presented) A method according to claim 17 comprising providing a plurality of non-nestable locking mechanisms.

24. (Previously Presented) A method according to claim 17 wherein the scheduler is called at the end of an interrupt service routine which is caused to run on the computing device.

25. (Canceled).

26. (Currently Amended) A method according to claim [[25]] 17 wherein the algorithm is configured to delegate memory management to a replaceable memory model configured in dependence upon a configuration of the computing device.

27. (Previously Presented) A method according to claim 26 wherein the memory model is configured to run in either pre-emptible or non-preemptible modes.

28. (Currently Amended) A method according to claim 27 wherein [[a]] the mutex is configured to protect the module from running in the pre-emptible mode.

29. (Previously Presented) A method according to claim 17 wherein the scheduler is included in a kernel.

30. (Previously Presented) A method according to claim 29 wherein the kernel comprises a microkernel or a nanokernel and wherein the threads are, respectively, microkernel or nanokernel threads.

31. (Currently Amended) A method according to claim 29 wherein the scheduler is called ~~each time~~ in response to the kernel [[is]] being unlocked.